**Práctica 2. Programación Avanzada**

Ignacio Gomís, Juan Pablo Uriol

**En esta práctica hemos utilizado la librería Google Test para crear pruebas de unidad que garantizan que las optimizaciones aplica cadas no afecten al correcto funcionamiento correcto de los algoritmos. Están el el fichero sort\_unittests.cpp. El fichero sort\_test2.cpp que pide en enunciado lo hemos sort\_main.cpp para evitar potenciales confusiones con el fichero de las pruebas de unidad.**

**A continuación exponemos los resultados obtenidos:**

Ejecución de tiempos inicial:

CountSort: 0.000363358 segundos por ejecución

RadixSort en Vectores: 0.002251087 segundos por ejecución

RadixSort en Listas: 0.0105267 segundos por ejecución

Cambios aplicados en los algoritmos:

1)

Cambiado Insert por Slice

Algoritmos Afectados: RadixSort en Listas

Mejora de tiempos: 38.95% de mejoría

2)

Cambiada condición del bucle de i<log10(max)+1 por i<=log10(max)

Algoritmos afectados: RadixSort en Listas (esto ya se había aplicado previamente en Radix Sort vector)

Mejora de tiempos: 21.19% de mejoría

3)

Descontado el tiempo de cálculo del numero aleatorio:

Algoritmos Afectados: Todos

Mejora de tiempos:

CountSort: 28.15% de mejoría

Radixsort Vector: 4.87% de mejoría

Radixsort Lista: 4.13% de mejoría

Tiempos en este punto:

CountSort: 0.000261083 (Mejora actual del 28.15%)

Radixsort Vector: 0.00214142 (Mejora actual del 4.87%)

Radixsort Lista: 0.00485591 (Mejora actual del 53.87%)

Cambio de ordenador, tiempos con mismo código:

CountSort: 0.00048392 s

Radixsort Vector: 0.00396 s

Radixsort Lista: 0.00943 s

4)

Cambio de base de los buckets a Base 16:

Algoritmos Afectados: RadixSort, ambos

Mejora de tiempos:

Radixsort Vector: 84.19% de mejoría

Radixsort Lista: 53.01% de mejoría

5)

Desenrollado de bucles del vector a ordenar:

Algoritmos Afectados: CountSort

Mejora de tiempos: 5.45% de mejoría

6)

Desenrollado de bucles del vector auxiliar, con try-catch para evitar errores:

Algotirmos Afectados: Count Sort

Mejora de tiempos: Empeora un 90.74%

Cambios revertidos

7)

Realizar un reserve para cada bucket

Algoritmos afectados: RadixSort Vector

Mejora de tiempos: Empeora del 29.3%

8)

Realizar un reserve solo para el bucket donde se concatena todo el vector

Algoritmos afectados: RadixSort Vector

Mejora de tiempos: Empeora del 0.62%

9)

Usar un move assignment para mover de los vectores auxiliares al vector:

Algoritmos Afectados: RadixSort, ambos

Mejora de tiempos:

Radixsort Vector: 1.43% de mejoría

Radixsort Lista: 6.95% de mejoría

10)

Empleo de ++i en lugar de i++

Algoritmos Afectados: Todos

Mejora de tiempos:

CountSort: Empeora un 1.94%

Radixsort Vector: 1.5% de mejoría

Radixsort Lista: 0.52% de mejoría

11)

Usar Inline en las funciones digito y concatenar vector:

Algoritmos Afectados: RadixSort, ambos

Mejora de tiempos:

Radixsort Vector: Empeora un 1.4%

Radixsort Lista: Empeora un 0.7%

Mejores tiempos en este ordenador:

CountSort: 0.00045753 s (Mejora total de 5.45%)

Radixsort Vector: 0.00060775 s (Mejora total de 84.65%)

Radixsort Lista: 0.00410202 s (Mejora total de 56.5%)

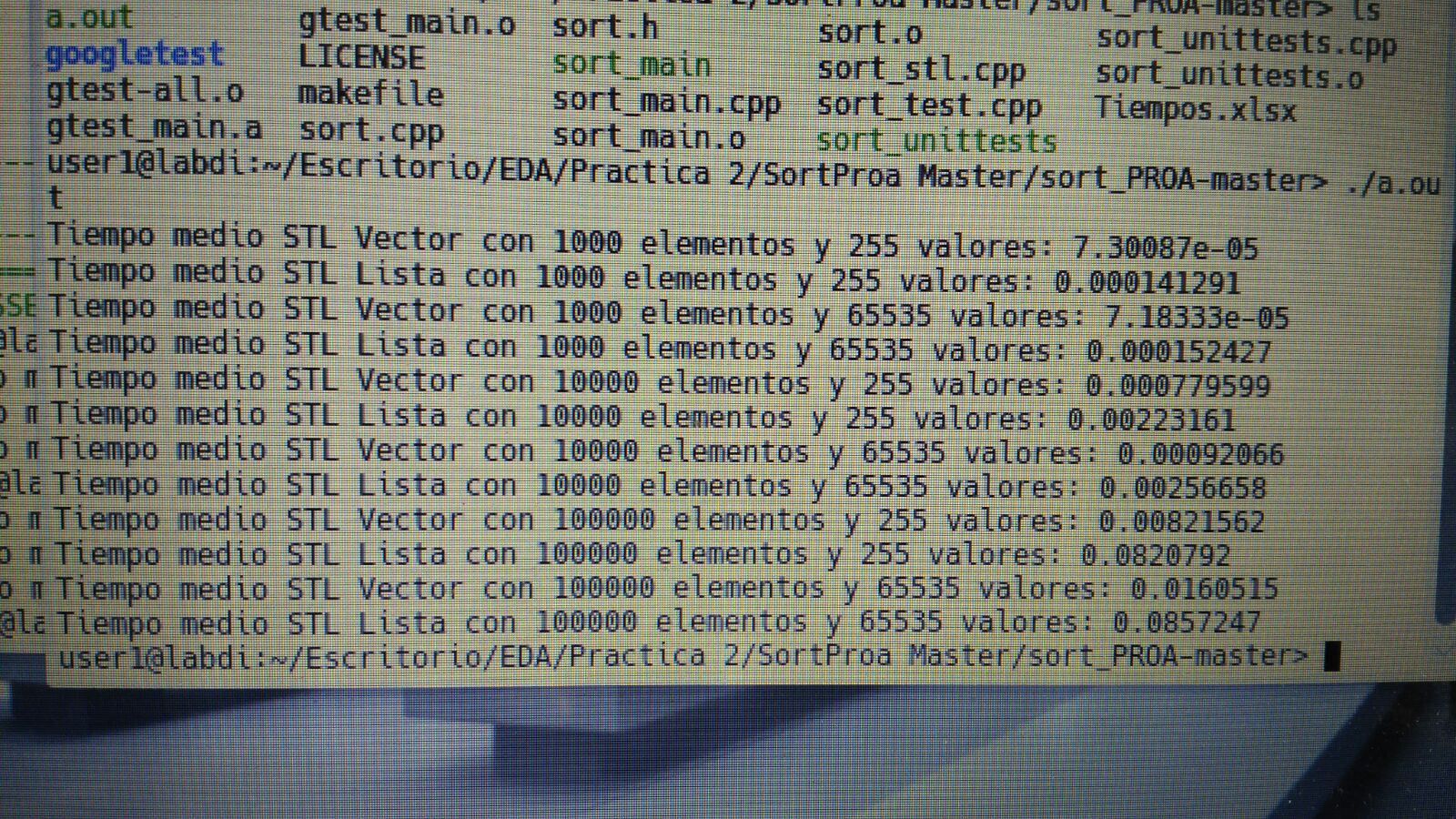
Tiempos estimados en el ordenador de juan pablo:

CountSort: 0.00024685 s (Mejora total de 32.06%) //Podría ser erróneo debido a una corrección intermedia en el algoritmo para soportar el 0

Radixsort Vector: 0.00032871 s (Mejora total de 85.39%)

Radixsort Lista: 0.00211217 s (Mejora total de 79.93%)

Ejercicio 3)



De una manera más comoda:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Vector | 255 | 65535 |  | Lista | 255 | 65535 |
| 1000 | 0.0000730087 | 0.0000718333 |  | 1000 | 0.0001412910 | 0.0001524270 |
| 10000 | 0.0007795990 | 0.0009206600 |  | 10000 | 0.0022316100 | 0.0025665800 |
| 100000 | 0.0082156200 | 0.0160515000 |  | 100000 | 0.0820792000 | 0.0857247000 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | Nuestra marca: | 0.00060775 |  |  | Nuestra marca: | 0.00410202 |

Podemos observar que las listas tienden a ser más lentas que los vectores, que el tamaño de los vectores ordenados también afecta a la ejecución, aunque de manera poco significativa, resultando realmente apreciable con grandes vectores.

La conclusión más importante, es que el tiempo se puede observar que incrementa linealmente según el tamaño del vector.

Por ello, creemos que la función que emplean para ordenar es una de las estudiadas en esta asignatura, probablemente LSD RadixSort, debido a que se puede implementar para cualquier tipo de dato mientras que count sort necesita que sea un listado de enteros (o que lo contenga)